Unit-3

header — Send a raw HTTP header. header() must be called before any actual output is sent, either by normal HTML tags, blank lines in a file, or from PHP. It is a very common error to read code with include(), or require(), functions, or another file access function, and have spaces or empty lines that are output before header() is called.

General syntax

header(string, replace, http_response_code)

Table 1: parameters of header

Parameter	**Description**
string	Required. Specifies the header string to send
replace	Optional. Indicates whether the header should replace previous or add a second header. Default is TRUE (will replace). FALSE (allows multiple headers of the same type)
http_response_code	Optional. Forces the HTTP response code to the specified value (available in PHP 4.3 and higher)

Parameters

String : the header string

There are two special-case header calls. The first is a header that starts with the string "***HTTP/***" (case is not significant), which will be used to figure out the HTTP status code to send. For example, if you have configured Apache to use a PHP script to handle requests for missing files (using the ***ErrorDocument*** directive), you may want to make sure that your script generates the proper status code (example 1)

Example 1: Script producing status code

```
<?php
header("HTTP/1.0 404 Not Found");
?>
```

The second special case is the "Location:" header. Not only does it send this header back to the browser, but it also returns a ***REDIRECT*** (302) status code to the browser unless the ***201*** or a ***3xx*** status code has already been set. (example 2)

Example 2: redirecting

```
<?php
header("Location:http://www.example.com/"); /*Redirect browser*/
/*Make sure that code below does not get executed when we redirect.*/
exit;
?>
```

Replace

The optional ***replace*** parameter indicates whether the header should replace a previous similar header, or add a second header of the same type. By default it will replace, but if you pass in **FALSE** as the second argument you can force multiple headers of the same type. For example:

Example 3: using replace parameters

```
<?php
header('WWW-Authenticate: Negotiate');
header('WWW-Authenticate: NTLM', false);
?>
```

http_response_code

The last argument is HTTP response code. While $replace argument maybe not so useful, $http_response_code might come handy, because it allows to select method of redirection. We will be interested in 3xx response codes as they are responsible for redirections.

- 301 – Moved Permanently. This and all future requests should be directed to the given URI.
- 302 – Moved Temporarily. Usually browsers use it as 303 redirection
- 303 – See Other. Resource you are looking for can be found under different URL
- 304 – indicates the resource has not been modified since last requested.
- 305 –Use Proxy
- 306 – Switch Proxy
- 307 – Temporary Redirect
- 308 – Resume Incomplete

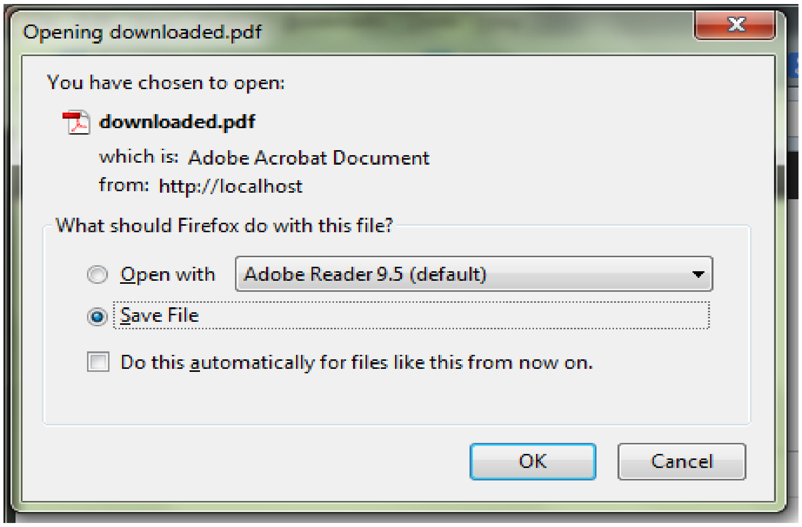
Example 4 : If you want the user to be prompted to save the data you are sending, such as a generated PDF file, you can use the [» Content-Disposition]() header to supply a recommended filename and force the browser to display the save dialog.

Example 4 :

```
<?php
// We'll be outputting a PDF
header('Content-type: application/pdf');

// It will be called downloaded.pdf

header('Content-Disposition: attachment; filename="downloaded.pdf"');
// The PDF source is in original.pdf
readfile('original.pdf');
?>
```

The header() function allows you send a custom header command to the web browser.

Output of example4



You have chosen to open:

downloaded.pdf

which is: Adobe Acrobat Document

from: http://localhost

What should Firefox do with this file?

○ Open with Adobe Reader 9.5 (default)

● Save File

□ Do this automatically for files like this from now on.

The main thing to remember with the header function is that it must be the first thing that sends output in you script. If it is not, you will receive the "headers already sent" error. The output can include error messages or simply a blank line before your initial <?php statement.

Examples

Here are some valid PHP redirections:

header("Location: http://google.com");

header("Location: pages/page.php");

header("Location: http://google.com", **true**, 303);

header("Location: http://google.com", **true**);

Remember also that headers MUST be sent before any output is started. There are no exceptions from this rule, in fact you cannot even sent a blank line to the output, executing code like this:

text

<?php header("Location: ditio.net"); **?>**

will throw warning *"Cannot add/modify header information – headers already sent by…"*. To avoid such situation you can use output buffering or just make sure that nothing was sent to the client.

PHP and HTTP Authentication

PHP can use the already existing authentication form from the Apache web server. It is possible to use the header() function to send an *"Authentication Required"* message to the client browser causing it to pop up a Username/Password input window. Once the user has filled in a username and a password, the URL containing the PHP script will be called again with the predefined variables *PHP_AUTH_USER*, *PHP_AUTH_PW*, and *AUTH_TYPE* set to the user name, password and authentication type respectively. These predefined variables are found in the $_SERVER and *$HTTP_SERVER_VARS* arrays.

A typical transaction between an HTTP client and an HTTP server might comprise the following steps:

- the client requests a page that requires authentication but does not provide a user name and password; typically this is because the user simply entered the address or followed a link to the page
- the server responds with the 401 ("Unauthorized") response code, including the required authentication scheme and the authentication realm
- at this point, the client will present the authentication realm (typically a description of the computer or system being accessed) to the user and prompt for a user name and password; the user has the option to cancel at this point
- once the user has supplied a user name and password, the client adds an Authorization header (with value base64encode(username+":"+password)) to the original request and re-sends it
- in this example, the server accepts the authentication and the page is returned; if the user name is invalid or the password incorrect, the server might return the 401 response code and the client would prompt the user again.

How HTTP Authentication Works

Figure 1: below shows the interaction between a web browser and a web server when a request is challenged The browser sends a request for a resource stored on the server. The server sends back a challenge response with the status code set to 401 Unauthorized, and the header field WWW-Authenticate. The WWW-Authenticate field contains parameters that instruct the browser on how to meet the challenge. The browser may need to prompt for a username and password to meet the challenge. The browser then resends the request, including the Authorization header field that contains the credentials the server requires.

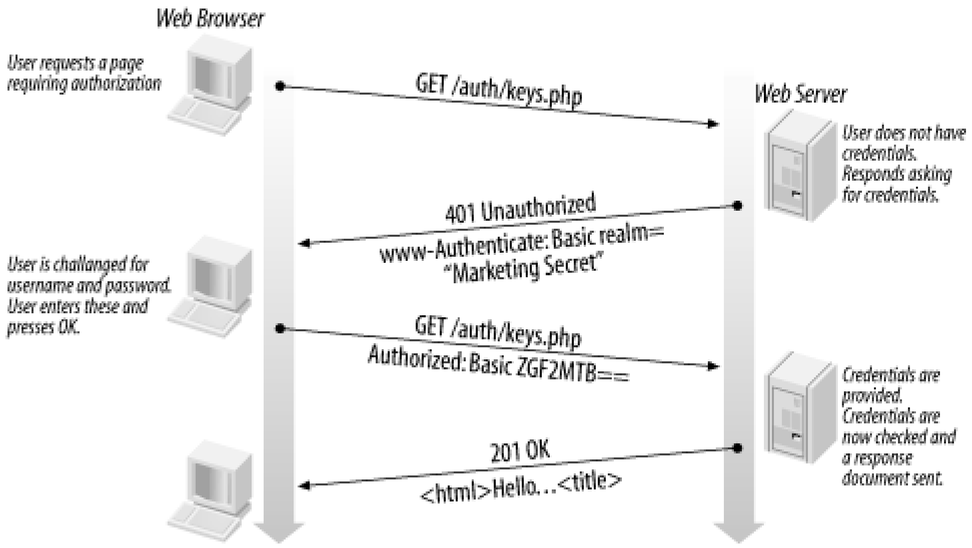

Web Browser
User requests a page requiring authorization
GET /auth/keys.php
Web Server
User does not have credentials. Responds asking for credentials.
401 Unauthorized
www-Authenticate: Basic realm= "Marketing Secret"
User is challanged for username and password. User enters these and presses OK.
GET /auth/keys.php
Authorized: Basic ZGF2MTB==
Credentials are provided. Credentials are now checked and a response document sent.
201 OK
<html>Hello...<title>

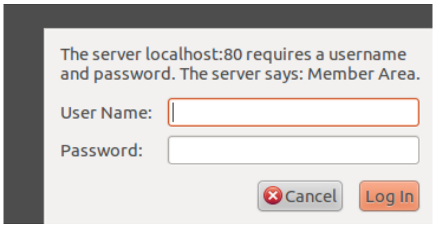
Figure 1: The sequence of HTTP requests and responses when an unauthorized page is requested

Example 5***: Using HTTP authentication with a PHP script***

```
<<?php
If (!isset($_SERVER['PHP_AUTH_USER']) ||
!isset($_SERVER['PHP_AUTH_PW']))
{
header('WWW-Authenticate: Basic realm="Member Area"');
header("HTTP/1.0 401 Unauthorized");
echo "Please login with a valid username and password";
exit;
}
else
{
echo "You entered a username of : ".$_SERVER['PHP_AUTH_USER'];
echo "  and a password of : ".$_SERVER['PHP_AUTH_PW'];
}
?>
```

The WWW-Authenticate header field contains the *challenge method*, the method by which the browser collects and encodes the user credentials. In the example the method is set to Basic. The header field also contains the name of the realm the authentication applies to.

The realm is used by the browser to label usernames and passwords and is displayed when credentials are collected; Figure 2 shows the realm ***Member Area***. A browser can automatically respond to a challenge if the browser has previously collected credentials for the realm. The browser stores authentication credentials for each realm it encounters until the browser program is terminated. Once the browser has collected the credentials, it resends the original request with the additional Authorization header field. The **Example 5**: shows a request containing encoded credentials in the Authorization header field.

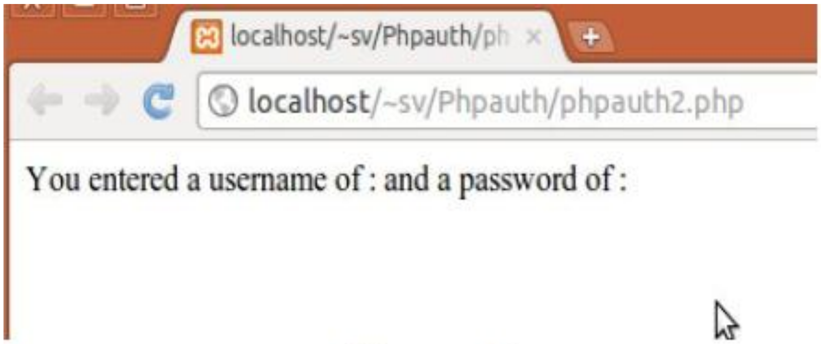


The server localhost:80 requires a username and password. The server says: Member Area.

User Name:

Password:

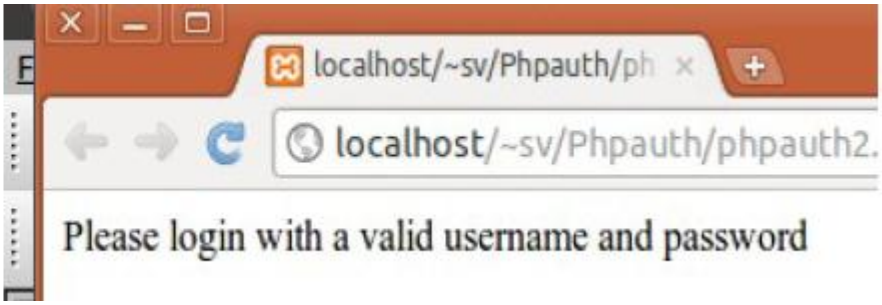
Figure 2: Request for authorization



You entered a username of : and a password of :

Figure 3

If the user clicks the login button without entering data , he'll see the *Figure 3*:



Please login with a valid username and password

If the user clicks the cancel button without entering data , he'll see the *Figure 4*

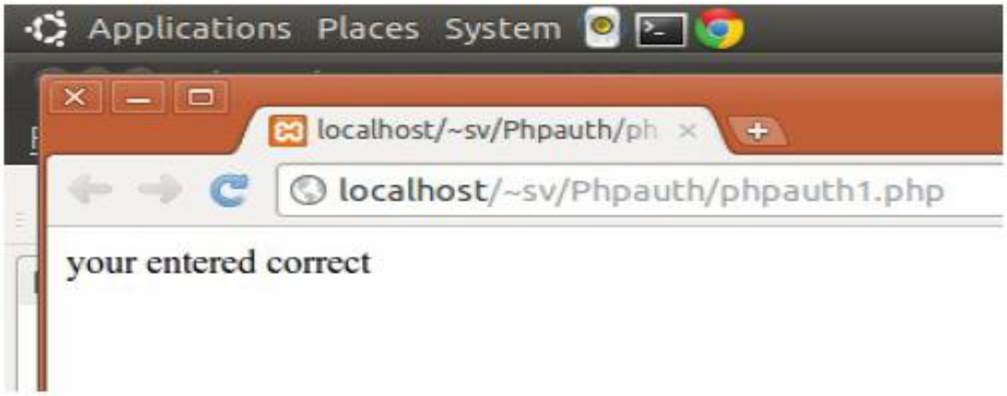
Storing username and password in the PHP script itself

The **example 6**: validates the username and password retrieved from an authentication prompt. This is given inside the PHP coding itself. Instead of this you can store username and password in a data base and check whether the user entered information is matching with that of present in the database.

Example 6: Checking the values returned from the authentication prompt

```
1. <?php
2. $username='sujatha';
3. $password='rangarajan';
4. if(!isset($_SERVER['PHP_AUTH_USER'])||
   !isset($_SERVER['PHP_AUTH_PW']))
5. {
6.     header('WWW-Authenticate: Basic realm="Member Area"');
7.     header("HTTP/1.0 401 Unauthorized");
8.     echo "you must enter a username and password combination";
9.     exit;
10.     }
11.     elseif (strcmp($_SERVER['PHP_AUTH_USER'],$username) !==0
12.      || strcmp($_SERVER['PHP_AUTH_PW'], $password) !==0)
13.      {
14.       header('WWW-Authenticate: Basic realm="Member Area"');
15.       header("HTTP/1.0 401 Unauthorized");
16.     echo "your username and password combination was
   incorrect";
17.        exit;
18.        }
19.        echo (" your entered correct");
20.     ?>
```

In the above code line 4 checks for the authentication of the user and their password which are allocated in the line 2 & 3.



your entered correct

If the values are not entered properly then the message in line 8 will be displayed on the browser.

Line 11 and 12 using the strcmp function checks for the exact match of the input values and the values defined in line 2 and 3 and if both are same then the line 19 is executed else line 16 is executed. On executing line 19 we can place allow the user to enter our web page by placing a hyper link there or by redirecting it. When the data is entered correctly the **Figure 5** is displayed.

Storing username and password in a database

Instead of storing the username and password in PHP script the values are stored in database and we are going to check. When you store the data in program it is not dynamic. Not only that we can check only against a single user. But suppose we want to allow our web page to be accessed by many authenticated user then the best choice is to store the data in the database and check the authentication. To do follow the steps

1) Create the data base
2) create the table
3) insert the user details into table.
4) Write PHP coding so that connectivity with database is formed and data is extracted and checked against login form for authentication.

```
mysql>create database user;
Query OK, 1 row affected (0.01 sec)

mysql> use user;
Database changed

mysql> insert into users values('sujatha',MD5('secret'));
Query OK, 1 row affected (0.00 sec)

mysql> select * from users;
+----------+----------------------------------+
| username | password                         |
+----------+----------------------------------+
| sujatha  | 5ebe2294ecd0e0f08eab7690d2a6ee69 |
+----------+----------------------------------+
1 row in set (0.00 sec)
```

Now that you've created the table. You've entered the MD5 function in INSERT command to provide an extra layer of security. Even if a malicious user can not encode this password.

First we write a config.php program which has the database information.

Example 7: Config.php

```
//config.php
<?php
$host="localhost";
$db_user="root";
$db_pass="";
$db_name="user";
?>
```

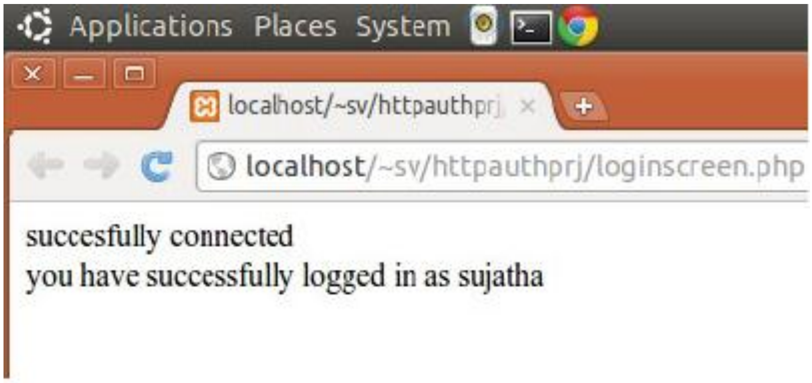
Next we write the program which gets input from http_authentication screen and verify the data with the database value and if both are same it allows the user to login.

Example 8: loginscreen.php

```
//loginscreen.php
<?php
require_once 'config.php';
 if(!isset($_SERVER['PHP_AUTH_USER']) ||
!isset($_SERVER['PHP_AUTH_PW']))
{
    header('WWW-Authenticate: Basic realm="Member Area"');
    header("HTTP/1.0 401 Unauthorized");
    echo "you must enter in a username and password combination";
    exit;
      }
      $uname=$_SERVER['PHP_AUTH_USER'];
      $pass=$_SERVER['PHP_AUTH_PW'];
      $dbhandle=mysqli_connect($host,$db_user,$db_pass,$db_name)
 or die("unable to connect");
      echo "succesfully connected <br>";
      $query="select * from users where username = "."'$uname'" ." and
password = MD5('".$pass."')";
      $st_table=mysqli_query($dbhandle,$query)
  or die("unable to execute".$query);
      if(!$row=mysqli_fetch_array($st_table))
      {
          header('WWW-Authenticate: Basic realm="Member Area"');
          header("HTTP/1.0 401 Unauthorized");
          echo "your username and password are incorrect";
          exit;
23.        }
24.        echo "you have successfully logged in as ".$row['username'];
25.        ?>
```

Line 6 prompt the user with login screen and if he enters erroneous data than line 7 is responsible for displaying unauthorized information and asks user to login with proper data. In line 11 and 12 we extract the value entered by user in local variable $uname and $pass. Line 13 will check for database connectivity. In line 15 the sql query is used to check database whether the username and password entered by the user in the login screen match with that of database fields. In line 17 checks whether the query resulted any row. If successful this means the user detail is present in database else the again the user is prompted with login screen by line 19 through 22.

The output of the program on successful login is given in **Figure 6**:


Applications Places System
localhost/~sv/httpauthprj
localhost/~sv/httpauthprj/loginscreen.php

```
succesfully connected
you have successfully logged in as sujatha
```

Figure 6

Security

we can tighten the security by providing security to the directories in which we have written our web site and its related programs. A way of creating this security is through creation of ***.htaccess*** file in the same web directory as the scripts. This file tells Apache to require a user to authenticate it before it returns any of the information in that directory.

To do this write the following information in **.htaccess** file and keep it in the directory for which you need security. In this case I have created a folder called pass under /home/sv

The use the following command for rights

chmod 777 /home/sv/pass

Then type the following to create new password file as such:

sv@ubuntu:~$ ***htpasswd -c /home/sv/pass/password radha***

Adding password for radha.

New password:

Re-type new password:

To create a .htaccess file open a text file (notepad or gedit) and type the following:

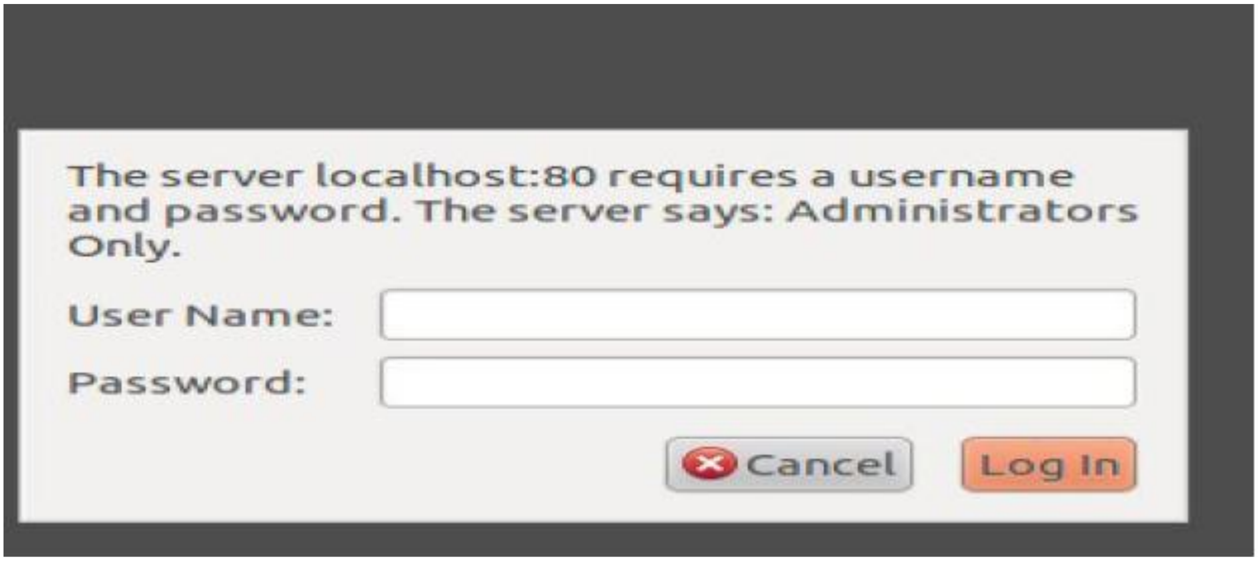
Example 9 : .htaccess

```
AuthType Basic
AuthName "Administrators Only"
AuthUserFile /home/sv/pass/password
Require valid-user
```

You have to store it without file name but with extension only as (.htaccess) and this has to be stored in the directory for which we are going to give more security. Requesting a directory or subdirectory where this file is stored causes the prompt in Figure 7 to display in Firefox or any browser.

Failure to supply a correct username and password causes the warning as in

Figure 8



The server localhost:80 requires a username and password. The server says: Administrators Only.

User Name:

Password:

Basic Authentication with .htpasswd in Xampp (Windows)

The basic http authentication created with `.htaccess` and `.htpasswd` files works pretty well on online real server. However implementing them locally on our Xampp server (on Windows) may be a little problematic.

First of all, we need 2 files need to be created, `.htaccess` and `.htpasswd`. All the user passwords are stored inside the `.htpasswd` file.

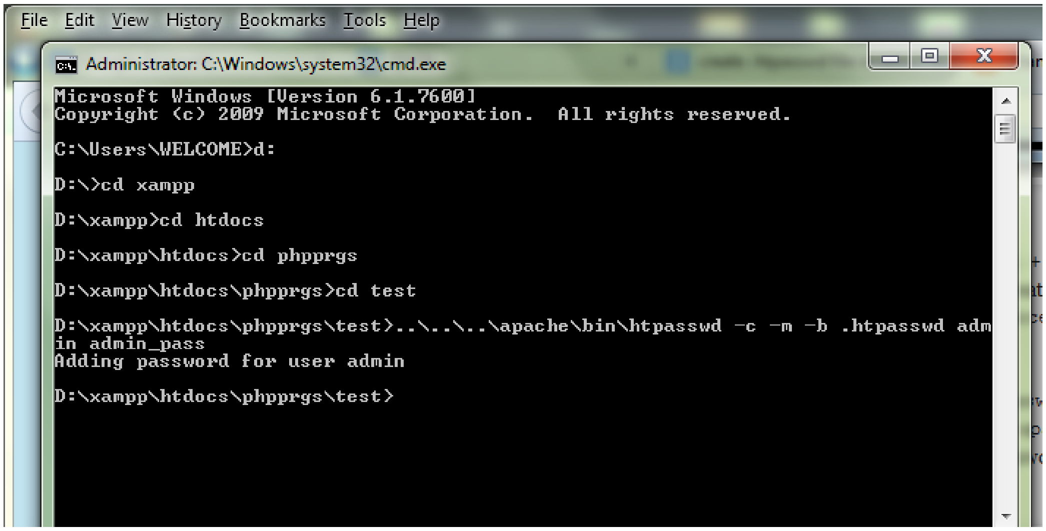
Open the notepad and type the following: Name the file as .htaccess (extension only no filename) and save it in the folder for which we want the security.

```
AuthType Basic

AuthName "my protected area"

AuthUserFile d:\xampp\htdocs\phpprgs\test\.htpasswd

Require valid-user
```

In this case an index.php file is created inside the folder test. So the file has to be stored inside the test folder

Our working folder is "test”. We created the `.htaccess` inside that only.

Next we would create the `.htpasswd` file with the help of command '`htpasswd`', the corresponding windows executable file '`htpasswd.exe`' resides in `xampp\apache\bin` folder. So, check the next screenshot to understand how we create the password file required for authentication.

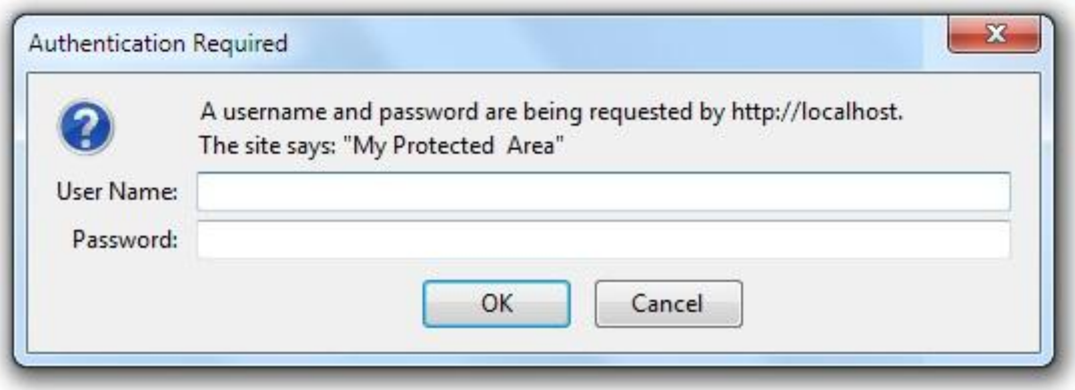

File Edit View History Bookmarks Tools Help
Administrator: C:\Windows\system32\cmd.exe

```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\WELCOME>d:

D:\>cd xampp

D:\xampp>cd htdocs

D:\xampp\htdocs>cd phpprgs

D:\xampp\htdocs\phpprgs>cd test

D:\xampp\htdocs\phpprgs\test>..\..\..\apache\bin\htpasswd -c -m -b .htpasswd adm
in admin_pass
Adding password for user admin

D:\xampp\htdocs\phpprgs\test>
```

We gave the command "`htpasswd -c -m -b .htpasswd admin admin_pass`". "`admin`" is going to be the username and "`admin_pass`" would be our password.

`-c` : create a new file
`-m` : MD5 encryption is enforced
`-b` : Use the password given at the command prompt

Check the third line where the path of the password file is clearly mentioned. Now when we try to go to `http://localhost/phpprgs/test`, a prompt appears asking for user id and password. If valid id and password given, then we are redirected to `index.php`.



A username and password are being requested by http://localhost.
The site says: "My Protected Area"

Use of Including files

Always store the include files which are having very important information like database connectivity, password etc with extension .php only instead of .inc. Because .inc allows the user to see the PHP source code. But if it is stored with .php the user cannot see the source.

Storing passwords in Database.

Always store the passwords in the database after encoding them. There are two ways in which you can encode 1) using **MD5** and 2) using **sha1**

<u>using MD5 :</u>

This returns a 128-bit string. You can use it while inserting data into database like below.

mysql> insert into users values('sujatha',MD5('secret'));

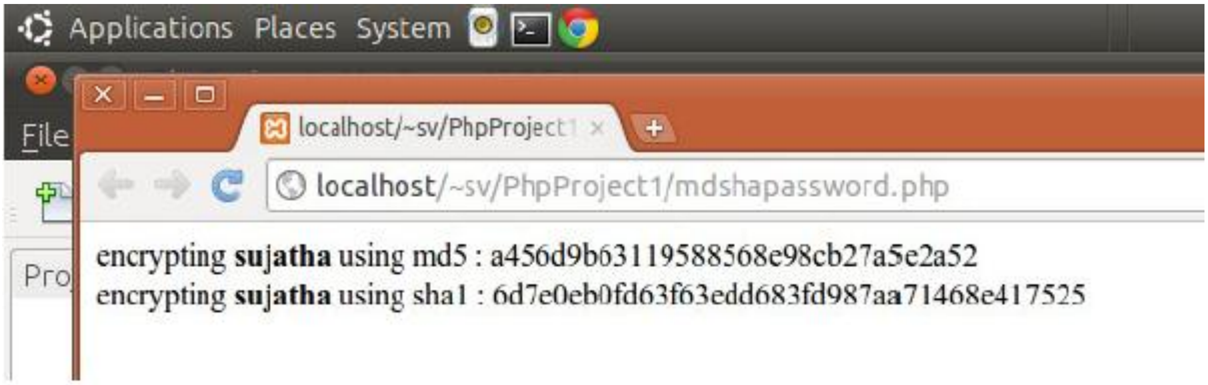
<u>using sha1:</u>

Returns a 16-bit string. This is more powerful ,advanced and complex than MD5. It is hard to break the code.

Example 10: use of md5 and sha1 for password entry

```
<?php
echo "encrypting <b> sujatha</b> using md5 : " . md5("sujatha");
echo "<br>";
echo "encrypting <b> sujatha</b> using sha1 : " . sha1("sujatha");
?>
```

The output for the above program is given in Figure 9


Applications
Places
System
File
Pro
localhost/~sv/PhpProject1
localhost/~sv/PhpProject1/mdshapassword.php

```
encrypting sujatha using md5 : a456d9b63119588568e98cb27a5e2a52
encrypting sujatha using sha1 : 6d7e0eb0fd63f63edd683fd987aa71468e417525
```

Figure 9: The output of MD5 and sha1

Validating and Error Handling

Validating User Input with javascript

On the client side, your best tool for validating data is JavaScript. JavaScript is different than PHP, because it's designed to execute in the user's browser instead of on the server. Because it executes in the client's computer, JavaScript is not allowed to access anything that could be a security risk, such as the local filesystem or network resources. JavaScript is primarily used in web pages. Although its name sounds like Java, it has no relationship to it. Some of the practical things you can do with JavaScript are checking fields and alerting the user to a problem before the data is submitted to the server

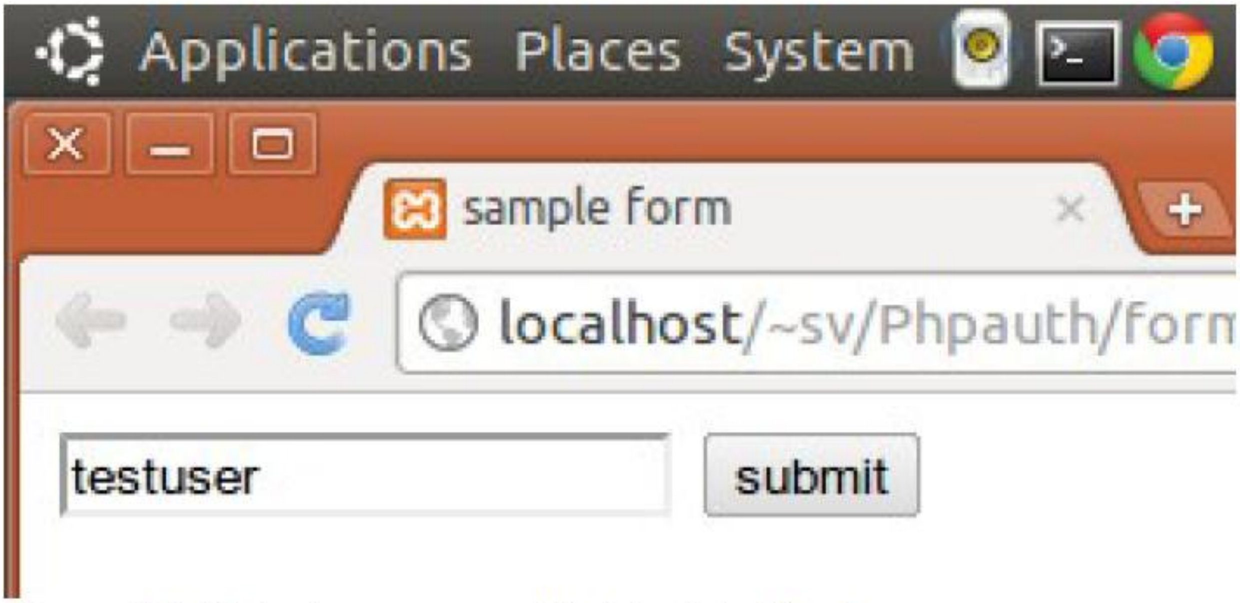
Example 11 : Building a form that validates its fields before submission.

```
//formvalidate.html
<script lang="JavaScript1.2" src="source.js">
 </script>
<html>
    <head>
        <title> sample form </title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    </head>
    <script language="JavaScript1.2">
        function check_valid(form)
              {
                  var error="";
                  error += verify_username(form.username.value);
                  if (error != "")
                 {
                  alert(error);
```

```
                return false;
               }
            return true;
            }
            </script>
        <body>
            <div>TODO write content</div>
            <form action="formvalidate.php" method="post"
                  onsubmit="return check_valid(this)" id="test1" name="test1">
                <input type="text" name="username">
                <input type="submit" value="submit">
            </form>
        </body>
    </html>

```

Example 12: javascript file : souce.js

```
function verify_username(strng)
{
  var error="";
  if(strng =="")
    {
      error = "you don't enter a username \n";
    }
    var illegal=/\W/;
    if((strng.length<6) || (strng.length>10))
      {
        error="the username is wrong length";
      }
      else if (illegal.test(strng))
        {
          error="the username contains illegal char";
        }
        return error;
}
```



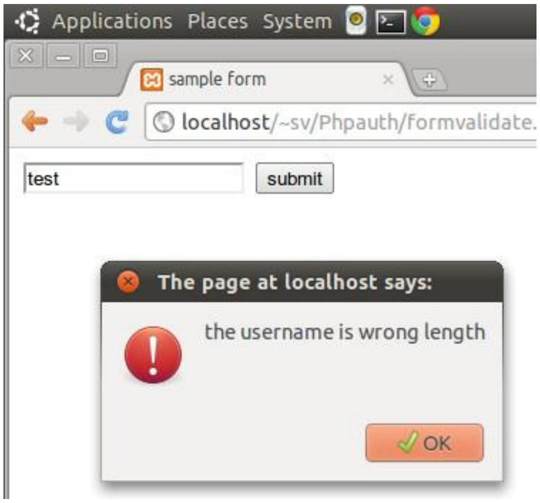
localhost/~sv/Phpauth/forn

testuser

Figure 10: Entering some valid data into the form

Line 1 in example 11 includes the javascript file “source.js”. This program is given as example 12. In line 12 the data from form control username (textbox) is sent for validation to function verify_username() which is written in example 12. The function returns the error message which is displayed as alert box on the screen. The html from line 24 call the function check_valid() which takes all the inputs in the controls of the form. This function is written in line 9 as javascript.

In *example 12* the function verify_username() takes the input which is mentioned in the form and checks in line 4 whether the user has entered any input or not. If not it returns the error message in line 6. Once the user enters the input line 9 checks for length of input and returns the appropriate error . Line 13 checks for illegal character which is defined in line 8. **\W** (allow letters, numbers, and underscores). Figure 11 and Figure 12 gives the error messages.



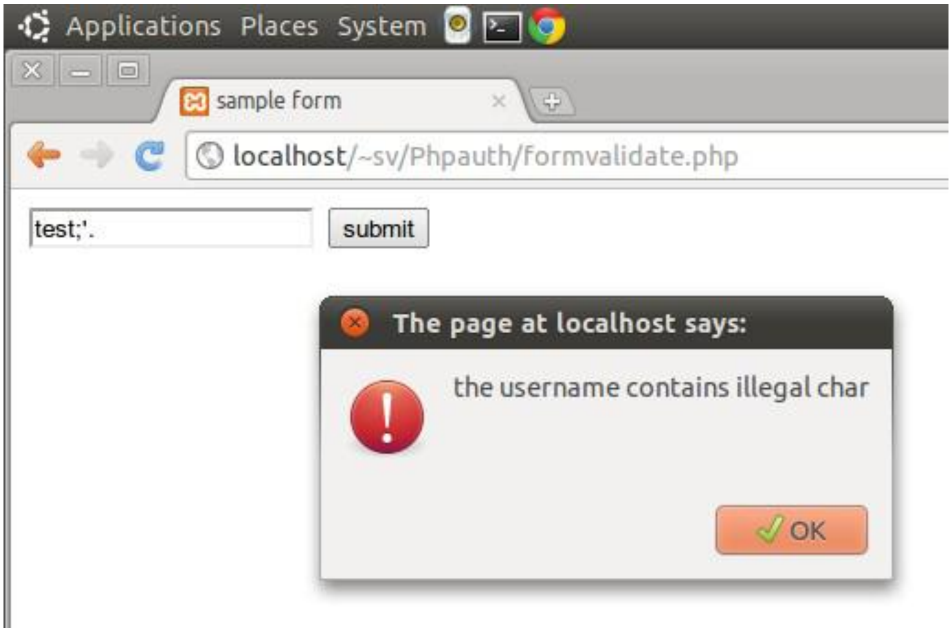
localhost/~sv/Phpauth/formvalidate.

test

The page at localhost says:

the username is wrong length

Figure 11 : The   javascript alert window that lists the validation problems



test;'.

The page at localhost says:

the username contains illegal char

Figure 12 : java script validation screen

Pattern Matching

Pattern matching allows you to build expressions that match strings using a specific matching syntax called a regular expression. Regular expressions allow you to perform searching tasks such as separating out a certain tag for an incoming text file, or validating user input such as email addresses.

The easiest way to use regular expressions in PHP is to use the PCRE (Perl-compatible regular expressions) extension. This extension is installed by default, so it should be part of your PHP environment. PHP also supports a style of regular expression matching functions called ereg that are older and less compatible than PCRE functions.

Regular Expressions

A regular expression is really just a string. The string uses a combination of special characters and literals to allow matching of other strings. For example, the following string describes an email address:

```
¥b[A-Z0-9._%-]+@[A-Z0-9._%-]+¥.[A-Z]{2,4}¥b
```

It does this by searching for:

1. Sequential alphanumeric and punctuation characters, which form the username (**[A-Z0-9._%-]+)**
2. The at symbol (@) (that separates the username and domain name)
3. A group of alphanumeric and punctuation characters, which forms the first part of the domain name (**[A-Z0-9._%-]+**)
4. A period, which separates the domain name from the extension (**\.**)
5. A two- to four-character alpha string, which signifies the top level domainfor example, com and net （[A-Z]{2，4}）

Table 2: The descriptors used in the regular expression are:

Regular expression	**meaning**
\b	A boundary point of a word
[aAbB]	One of anything inside the brackets: a， A， b， B
{2,4}	A total of between 2 and 4 of anything preceding the brackets
A-Z	Any letter between A and Z, such as A， B, and C
\.	A literal period
+	Match the preceding block one or more times

There are two types of characters in the regular expression string. Those that match themselves, such as the at (@) symbol, are called literals, meaning they literally match. The other type is called metacharacters, which describe matching by specifying repetition, ranges, and combinations within the expression.

Quantifiers

Quantifiers are metacharacters that specify how many times you wish to match the preceding pattern in a string. They are given in table 3.

Table 3: Quantifiers

Quantifiers	**meaning**
*	Zero or more
+	One or more
?	Zero or one
{ *num*}	Exactly *num* times
{ *num*,}	At least *num* times
{ *min*, *max*}	At least *min* but not more than *max* times

For example, the regular expression [a-f]?ex matches both alex and ex, but not ax.

Anchors

Anchors define a specific location for a match to take place.

To match the start of a line, the caret character (^) is used. To match the end of a line, the dollar character ($) is used. To match a string that begins with I, use the regular expression ^I.

Other anchors deal with word boundaries. Words are made up of consecutive letters, digits, and underscores. All other characters, such as spaces, punctuation, and newline characters, are word boundaries. To match a word boundary, the backslash b (¥b) character is used. To match everywhere that isn't a word boundary, the backslash capital B (¥B) character is used. Table 4 lists other word boundaries.

Table 4 Escaped word boundaries

Character	Anchor type
¥b	A word boundary
¥B	A nonword boundary
¥d	A single digit character
¥D	A single nondigit character
¥n	The newline character
¥r	The carriage return character
¥s	A single whitespace character
¥S	A single nonwhitespace character
¥t	The tab character
¥w	A single word character, alphanumeric and underscore
¥W	A single nonword character
^	match the start of a line
$	match the end of a line

Character classes

A character class allows you to group several characters together and work with them in a regular expression as though they were one character. Use the square brackets ([]) to group the characters together. For example, to match any alpha character twice:

```
[a-zA-Z]{2}
```

You can also use a negated character class, which selects the opposite of the character class by adding a caret (ˆ) character after the opening square bracket. Note that this is the only time that caret character doesn't represent an anchor. The following matches all nonalpha characters.

```
[ˆa-zA-Z]
```

Executing pattern matches in PHP

PHP uses a set of functions that start with `preg_` to perform regular expression operations on strings. These functions take a regular expression as a parameter in a string format. There are functions for doing a variety of operations on strings, including splitting them up and returning matching portions.

The regular expression string must be in Perl format, which specifies that the regular expression start with ’/ and end with /’. The regular expression goes between the single quote and slashes, as in ’/*regular expression*/’. Forward slashes in the expression must be escaped with a backslash. For example, **/home/example** becomes ’/¥/home¥/example/’.

To specify regular expression options such as case insensitivity, add the parameter to the end of the regex string after the last slash. These most common parameters are listed in table below.

Table 5 Regular expression characters

Regex character	**Meaning**
s	Dot matches all characters
i	Case insensitive
m	Match start and end of line anchors at embedded new lines in the search string

For example, use ’/abc/i’ to do a case-insensitive search of abc.

preg_match

The function preg_match is used to return all matches based on the supplied regular expression and string. The function value returned is true if a match is found. Its syntax is:

preg_match (*string pattern*, *string subject* [, *array groups*])

In Example 13 we search the string example to see if it has words that start with ple. Since the string doesn't start with ple, no results are returned.

Example 13 Using preg_match to return an array of matches that start with ple

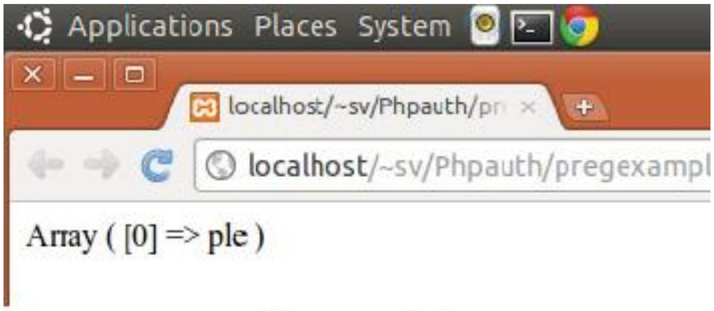
```
<?php
$subject = ″example″;
$pattern = ’/^ple/’;
preg_match($pattern, $subject, $matches);
print_r($matches);
?>
```

The output:

```
Array ( )
```

Example 14 Using preg_match to return an array of matches that has the string “ple” inside

```
<?php
$subject = ″example″;
$pattern = ’/ple/’;
preg_match($pattern, $subject, $matches);
print_r($matches);
?>
```



```
Array ( [0] => ple )
```

Fiaure 13

The following functions of javascript in example 15, example 16, example 17 uses regular expressions to check the input.

Example 15: JavaScript to check for password

```
// password verification
function verify_password (strng)
{
var error = "";
if (strng == "")
{
   error = "You didn't enter a password.\n";
}
    var illegalChars = /[\W_]/; // allow only letters and numbers
    if ((strng.length < 6) || (strng.length > 8))
   {
       error = "The password is the wrong length. It must be 6 to 8 characters.\n";
    }
    else if (illegalChars.test(strng))
    {
      error = "The password contains illegal characters.\n";
    }
    else if (!((strng.search(/(a-z)+/)) && (strng.search(/(A-Z)+/)) &&
(strng.search(/(0-9)+/))))
   {
```

```
20.        error = "The password must contain at least one uppercase letter, one
21.  lowercase letter, and one numeral.\n";
22.     }
23.  return error;
24.  }
25.
```

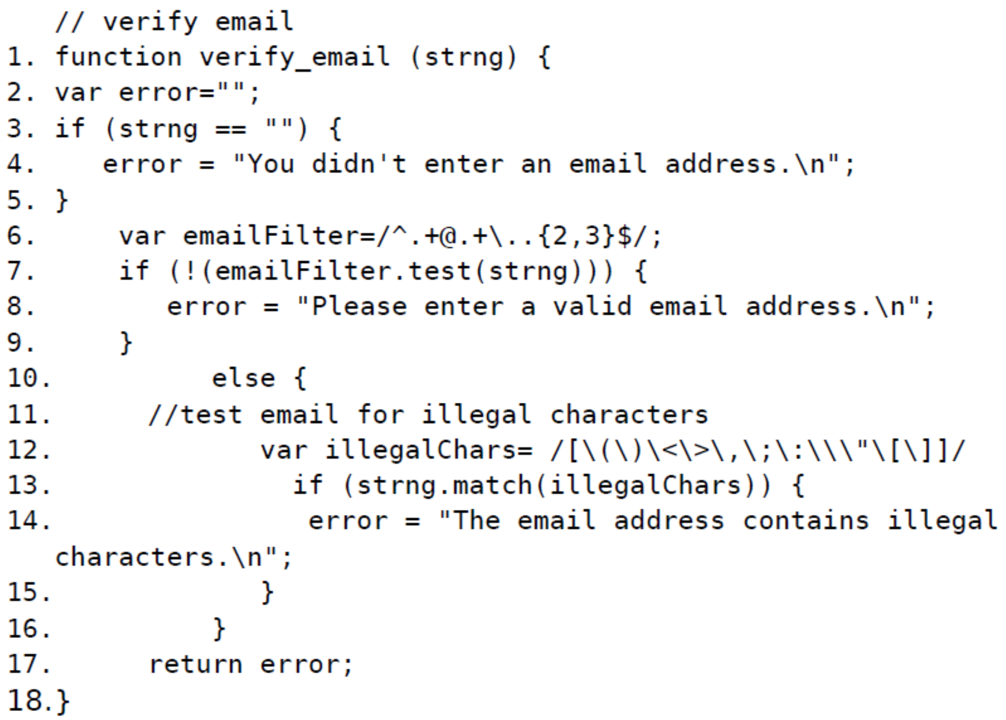
Explanation of the code: This code checks whether the user has entered valid characters in password field before sending the data to database in server side.

Line 8 we define the legal characters for password. This is done by ***/[\W_]/*** where the '***/***' forward slash in both end marks the boundaries and ***\W_*** indicates that it allows only numbers and letters. The use of ***[]*** square bracket is to group the letters and numbers. This is checked in line 13.

Line 9 checks for the length of the input.

Line 17 search for small case letters within /***(a-z)+***/ (I.e between a to z followed by more than that type and search for capital letters also by the string /***(A-Z)+***/ and search for the presence of numbers given by /***(0-9)+***/ and print the appropriate error messages.

Example 16: JavaScript to check for email entry



```
    // verify email
 1. function verify_email (strng) {
 2. var error="";
 3. if (strng == "") {
 4.    error = "You didn't enter an email address.\n";
 5. }
 6.     var emailFilter=/^.+@.+\..{2,3}$/;
 7.     if (!(emailFilter.test(strng))) {
 8.        error = "Please enter a valid email address.\n";
 9.     }
10.          else {
11.      //test email for illegal characters
12.             var illegalChars= /[\(\)\<\>\,\;\:\\\"\[\]]/
13.               if (strng.match(illegalChars)) {
14.                error = "The email address contains illegal
    characters.\n";
15.             }
16.          }
17.      return error;
18.}
```

Explanation of the above code:

Explanation of the code: This code checks whether the user has entered valid email id before sending the data to database in server side.

Line 3 checks whether the user has entered anything or not.

Line 6 we define the legal characters for email. This is done by ***/^.+@.+\..{2,3}$/*** where the '***/***' forward slash in both end marks the boundaries .

In line 12 illegal characters for an emailid is defined as **/[\(\)\<\>\,\;\:\\\"\[\]]/**

In this the characters are escaped with backslash. For example in the above code if it is like **\(** then the meaning is only square left bracket '**(** ' like wise the characters
[( ) < > , ; : \ “] are considered illegal. The validation of input string is done in the line 13.